Challenges of CAD Development for Datapath Design

Tim Chan, Design Technology, Intel Corp.
Amit Chowdhary, Design Technology, Intel Corp.
Bharat Krishna, Design Technology, Intel Corp.
Artour Levin, Design Technology, Intel Corp.
Gary Meeker, Design Technology, Intel Corp.
Naresh Sehgal, Design Technology, Intel Corp.

Index words: datapath, synthesis, automation, and generation

Abstract

In many high-performance VLSI designs, including all recent Intel® microprocessors, datapath is implemented in a bit-sliced structure to simultaneously manipulate multiple bits of data. The circuit and layout of such structures are largely kept the same for each bit-slice to achieve maximal performance, higher designer productivity, and better layout density. There are very few tools available to automate the design of a general datapath structure, most of which is done manually. Datapath design (from RTL to layout) very often takes a significant amount of human resources in a project. The design is becoming more complex and demanding as the clock frequency is reaching 1GHz, and the process technology is getting to 0.15um and below. Issues with signal integrity, as well as leakage current, are much more significant now as VCC and VT continue to be reduced and current density increases. Elaborate analyses on noise and power are needed for future designs, beyond the already complex timing, reliability, and functional correctness analysis tasks. The burden on CAD tools to support the high-performance microprocessor design is bigger than ever. This paper reviews the general approaches used in the industry to design datapaths from RTL to layout with the difficulties and issues encountered. We propose a new design workflow and a set of tools to improve overall designer productivity, while meeting all other constraints. A description of these tools to support the next generation of microprocessor design is also presented. Our proposed flow allows a designer to choose a design methodology ranging from a fully automated one to a custom one, to a flexible mix of the two. We present a new paradigm of early binding that considers the impact of circuit and layout during RTL design. We also strive to preserve RTL regularity during the circuit and layout design to improve time-to-market. Finally, we present some results on actual design blocks with the proposed tools and workflow, and we suggest future areas for further research.

I. Introduction

In most microprocessor design projects, the design team includes computer architects (particularly important for a design with new architecture), micro-architects (who determine the amount of hardware resources to be put in the chip and how the major data flow occurs), logic designers, circuit designers, and layout designers. Sometimes, these designers may have overlapping functions (for example, doing both circuit and layout design) depending on the experience level of the designers and the project management philosophy. Nevertheless, in general, designers of different disciplines need to communicate at different levels of design abstraction, and a design can only be completed when design data at different abstraction levels are consistent with each other and correct, meeting the design objectives.

1.1. Traditional Datapath Design Flow

For most high-performance microprocessors, the workflow for datapath design involves many labor-intensive steps [1, 2]. Logic designers and micro-architects determine the detailed features of hardware and the methods used to achieve particular functions. The number of pipe stages and which operations go with each pipe stage are also determined. These decisions are made with the help of bottom-up circuit

feasibility studies and some estimation tools for timing and area. The processes of developing the most appropriate computer architecture, micro-architecture, or RTL are also very involved, but they are beyond the scope of this paper. The starting point of the workflow is a partition of RTL coding for which timing and area estimations have been made and the results are within acceptance tolerance.

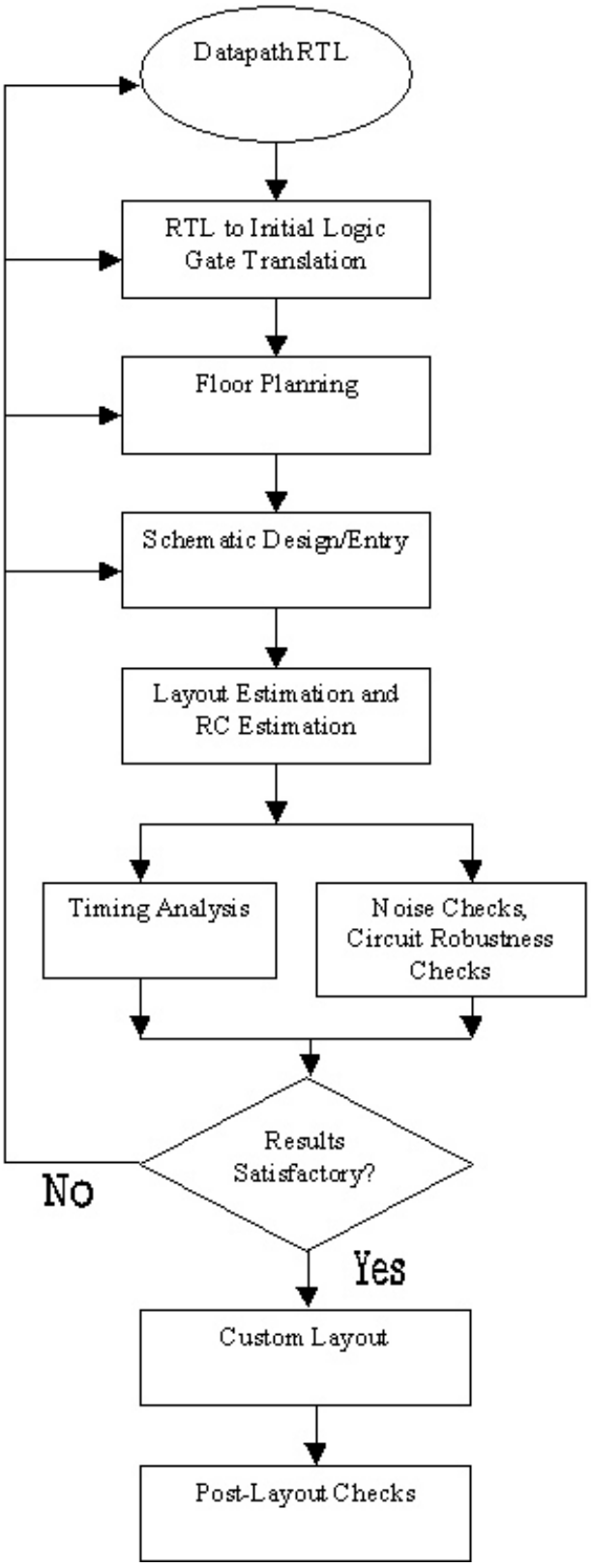

Figure 1: Datapath design flow

1.1.1. RTL to Initial Circuit Translation and Floor Planning

A circuit designer needs to study the functionality of the block first and then come up with the initial implementation plan that includes logic stages (without complete transistor sizing), floor planning, and circuit styles, based on different timing, area, and routing constraints. Depending on how experienced the designer is with the design techniques required for the block, feasibility studies and evaluations of a few design options are generally required to arrive at an implementation plan.

1.1.2. Schematic Design and Interconnect Estimation

The process of schematic design and interconnect estimation follows. As interconnect capacitance can be a significant portion of the total capacitance of a signal, a designer has to estimate the interconnects based on the assumptions made in his/her floor plan. In order to get a reasonable floor plan before actual layout is available, layout cell areas and pin/port locations are estimated. The "estimated" layout cells will be tiled according to the floor plan. With this rough datapath cell placement completed, interconnect lengths can be estimated based on the cell locations.

Circuit topology, transistor sizes, and floor plans will continue to evolve until a satisfactory design implementation is reached (though sometimes, the design specifications and external interfaces might also need to be modified). Once the schematic database for the datapath block is established, many checks and analysis can be performed.

1.1.3. Design Analysis

Next, the design with transistor sizes and interconnect RC's can be analyzed quite accurately for timing and many other circuit robustness requirements such as race conditions, noise tolerance, and long-term device and interconnect reliability. If the results of the analysis are not acceptable, the routing, cell placement, schematic design, RTL design or a combination of these will need to be modified.

1.1.4. Custom Layout and Post-Layout Checks

When the schematic with the corresponding estimated layout is satisfactory, layout can then be custom designed. Incorrect assumptions used in the estimated layout are corrected, and manual placement optimization is used.

When actual layout is completed, RC extraction is performed, and the RC netlists are merged back into the schematic netlist. At this point, all analyses of timing, noise, and circuit robustness are performed on the "accurate" netlist to verify that the design with actual layout data still meets the design requirements. If there is any problem found with the design, the design process is iterated until the design requirements are met.

With the top-down design process, even though the design requirements at one point are met, since other blocks in the chip might require design changes, the block needs to go through the Engineering Change Order (ECO) process. This is a formal procedure to communicate and implement changes to meet new requirements. In other words, the design process is reentered.

1.2. Issues with Traditional Datapath Design Flow

This design process mentioned above is quite top-down driven and sequential. From RTL to circuit and then to layout, each step makes a set of assumptions/estimations and provides more accurate information than the previous step. Each step of the design process also takes a significant amount of time to finish. As a result, poor estimations in early steps have very costly consequences due to the amount of time and effort required to make changes. In a large design project with a large team, the problems get multiplied many times over when poor estimates from one team member affect the design of other team members. As design specs are changing, implementations are not stable. Both become moving targets, and communication overhead increases substantially. As we have observed, large projects tend to require a long time for design convergence (i.e., when different pieces fit together and meet project requirements), and they have lower design productivity.

1.3. The Direction of Higher Levels of Automation

The accuracy of early estimations, and the turnaround time for the major design steps (e.g., RTL to circuit design) are very important elements when considering productivity in the design process. (Company culture, team maturity, design and management experience level are also part of the puzzle; however, these are not discussed in this paper.) Interestingly, more accurate early estimates and faster turnaround time can both be achieved with design automation. Automation that provides the correct result quickly can shorten the turnaround time, and it can also give more accurate estimations for the options that designers want to explore by quickly implementing these options. Though it is by no means easy to automate the design for high-speed complex microprocessor design using deep sub-micron technology, a lot of effort has al-

ready been made in the academic arena and by EDA companies. Automatic datapath cell layout generation and datapath block place and route tools are now commercially available. Compilation to datapath circuit-level netlist from hardware description language is also getting popular (in the less performance-critical designs such as chip set design). Over time, manual circuit and layout design techniques are digested by CAD developers who then formulate methods and heuristics to solve these design problems with CAD tools. However, this work is just not done soon enough to alleviate the burden of the designers for high-performance designs.

1.4. Structure of the Paper

The objective of this paper is to share our view of the high-performance datapath design problems and our ideas of what the solutions will look like, by providing some details of our work. Naturally, we don't have all the answers, but we believe that we have some good ideas about how these problems should be approached. It is hoped that this paper will stimulate readers to come up with more and better ideas.

The next section describes a more automated workflow compared to the workflow just described. The new workflow features automatic schematic generation from RTL and layout synthesis. The techniques of regularity extraction and how they are used in logic synthesis and schematic generation are discussed in the section on datapath logic synthesis. An efficient approach to design datapath schematics and to layout planning together is described in the section on datapath layout planning and placement. Accurate and efficient RC estimation is essential to various steps of the design process, and it is discussed in the section on the parasitic estimator. Datapath cell layout synthesis significantly reduces layout design resources for high-performance design, and it is discussed in the section on layout cell generation.

II. A More Automated Design Flow

A new workflow, which drastically improves productivity, is shown in Figure 2. It supports synthesis from RTL to layout, though with the understanding that datapath synthesis techniques will take time to mature. Designers input and user interfaces are essential to every step of the process.

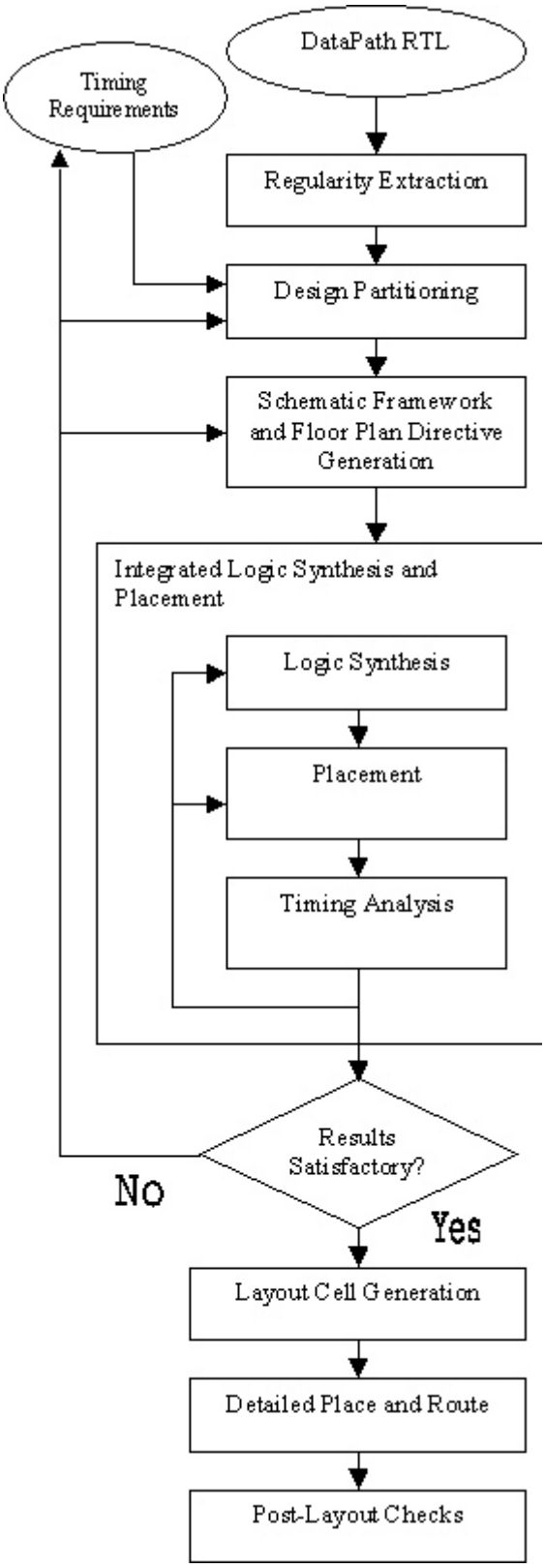

Figure 2: A more automated datapath design flow

The rest of this section describes the rationale behind each component of the design flow and gives the high-level expectations of these steps.

2.1. Regularity Extraction

In this workflow, regularity extraction is performed first to identify the repeated functionalities in RTL and to come up with optimal logic templates for logic synthesis at later stages. A good template needs to contain a few stages of logic (at least) to allow for the synthesis tool to perform optimization; however, it cannot be too large (containing too many logic functions) such that a lower level of regularity in circuit and layout cannot be exploited and therefore cause device density and performance to suffer as a result. In fact, the main reason for doing regularity extraction is the inability of the current synthesis tools to produce regular structures from RTLs for repeating functions. Secondly, with a logical netlist available from regularity extraction, designers can control the degree of regularity used in synthesis and modify the outcome of synthesis more easily.

2.2. Design Partitioning

This step is performed to identify what circuit style to use for different parts of the design. The commonly known circuit styles are static and dynamic. Normally, static is the first choice due to the robust nature of the style and the ease of design. However, in terms of speed, dynamic circuits are generally about 30% faster, and this style needs to be used when the speed of the circuit is critical. The price of using dynamic circuits is higher power consumption and greater design effort. As for high fanin logic, the use of dynamic circuits is more advantageous. Thus, a design partitioner is expected to estimate the timing performance of the datapath block with static circuits and single out the paths that are not meeting the performance requirements. Once some sections of the logic have been identified for dynamic circuit implementation, the logics going in as input to these dynamic circuits need to be considered as candidates for dynamic implementation as well, in order to ensure correct circuit functionality.

Also, it is expected that the current logic synthesis tools are not able to produce optimized results for complex special functions, such as a 32-bit adder (which involves a lot of special circuit techniques and fine-tuning). A datapath macro cell library (probably with special macro cell-sizing techniques) needs to be used to supplement the deficiency of current synthesis tools. As a result, the design partitioner needs to identify the logic functions that should be supported by a macro cell library (such as adders, register files, and comparators) and later target those functions for macro cell mapping and sizing.

2.3. Schematic Hierarchy and Floor Plan Directive Generation

Schematic hierarchy generation follows after design partitioning is done, and even though at this point no actual logic gate or transistors have been mapped, a schematic hierarchy with logic templates can be created. With schematics, circuit designers can proficiently modify the design partitioning and hierarchy for better synthesis results. Again, it is not expected that perfect results can be achieved by the design partitioner, and input from the designer is very crucial at this point. With regularity reflected in the hierarchical schematics, designers can modify the datapath cell placement directives (for placing cells into rows and bit columns) that are created by tools using heuristics.

2.4. Integrated Logic Synthesis and Placement System

Once the partitioner has been given input for synthesis and directives for placement, the integrated synthesis and placement phase is entered. The main reason an integrated system for synthesis and placement is needed is that doing logic synthesis without placement information does not give good enough results for future process technologies (0.15um or below). Transistor intrinsic delay continues to improve, and the average percentage of interconnect capacitance over the total node capacitance continues to increase. Interconnect delay has become an important component in very high-performance design, and the traditional wire load model used in control logic synthesis is not adequate for high-performance datapath synthesis. Placement information (in turn, RC information) needs to be available for the synthesis tool for correct sizing, buffering, signal repeating, and circuit topology choice.

2.5. Integrated Schematic Design and Layout Planning Environment

In the same spirit, designers need to be able to interact with schematics (outcome of synthesis), and placement needs to be integrated into the design tools. The tools have to efficiently support modifications of schematics and placement (RCs) by the designers, and be quickly able to communicate the changes among themselves to enable designers to see the effects of their changes (on timing, area, power, and noise, etc.).

2.6. Layout Cell Generation

When logic synthesis and global placement are completed, layout cells at the layout hierarchy assigned by the placement tool are then generated. A lot of meth-

odology definitions have to be completed before layout generation, such as power griding structures and usage of metal layers for cell pins and ports. Metal width and space requirements for reliability and noise concerns are also considered.

Layout cell generation is not the only way to create the bottom hierarchy of the layout. Library cells can also be used as in the traditional control logic layout synthesis. The layout quality of library cells is expected to improve as more effort has been put into library cells that are expected to be used by different projects. However, layout density might not be as good when compared to layout done with cell generation, since cell generation processes more devices together and has the opportunity to achieve better optimization.

2.7. Detailed Place and Route

After layout cells are generated, they can be used for detailed place and route (which is the process of generating DRC-clean placement and routing, based on the approximate (sometimes incomplete) results from global placement and routing. If global place and route are done well, it is expected that detailed place and route will only change the RC results by 5%. When a DRC-clean layout is completed, RC extraction can be performed, and all the necessary post-layout analysis can then be done with accurate RC information. The analyses normally include electrical rule checks, noise, timing violations, and setup and hold time checks (min-delay analysis).

Now that we have outlined the overall design flow, we focus on the details of the major design steps in the following sections.

III. Datapath Logic Synthesis

Logic synthesis, which transforms a design from RTL to circuit level, has been widely studied for control logic. Logic synthesis [5, 6] involves two steps: logic minimization followed by technology mapping to a user-specified library. Datapath circuits possess a very high degree of regularity that has to be preserved throughout the design process to achieve high density and performance. If the traditional logic synthesis approach based on logic minimization is used, then some regularity would be lost, resulting in inferior results. Therefore, an ideal datapath synthesis approach should first extract the regularity inherent in RTL descriptions prior to mapping the circuit to a desired technology. The extracted regularity results in a design hierarchy, which should be preserved to achieve high design quality as well as productivity.

We propose a novel methodology for logic synthesis of datapath circuits, where the datapath regularity is first extracted and then the circuit is mapped to a desired technology while preserving regularity. The input to our synthesis approach is an RTL description of a datapath circuit. Regularity in the circuit implies the existence of subcircuits, called templates, which have multiple instances in the circuit. Regularity extraction first identifies a sufficiently large set of templates and their instances, and then completely covers the circuit by a subset of these template instances. The template instances are then grouped to form datapath vectors. A schematic of the datapath is generated using these vectors and the boundary constraints on the I/O buses and signals. The schematic helps the designer in understanding the circuit and in making important decisions about or changes to the templates and vectors identified so far. The next step is to map the templates to static and dynamic logic as desired, thus resulting in efficient multi-technology designs. Finally, the mapped templates are sized according to the loading on the primary outputs of the circuit.

```
Module Example1

Inputs      a[3:0], b[3:0], c[1:0];
Inputs      sel1, sel2 sel3, sel4;
Outputs p[3:0];

begin main
    for i = 0 to 1 do
        f[i] = cond begin
                    [sel1]      b[i];
                    [sel2]      c[i];
                    []          0;
                    end;
    for i = 2 to 3 do
        f[i] = cond begin
                    [sel1]      b[i];
                    []          0;
                    end;

    /* increment f[3:0] to get g[3:0] */
    /* f[3:0] = g[3:0] + 1;                    */
    /* we use the following gate-level representation */
    /* of the incrementer for illustrating our approach */
    carry[1] = f[0] AND f[1];
    carry[2] = carry[1] AND f[2];
    carry[3] = carry[2] AND f[3];

    g[0] = NOT f[0];
    for i = 1 to 3 do
        g[i] = carry[i] XOR f[i];
    /* end incrementer*/

    for i = 0 to 3 do
        p[i] = cond begin
                    [sel3]      g[i];
                    [sel4]      a[i];
                    []          0;
                    end;
end main;
```

Figure 3: HDL description of a small datapath circuit used to illustrate our synthesis approach

We describe below in detail the various steps in our synthesis methodology of datapath circuits. We explain our methodology with the aid of the circuit in Figure 3 (the corresponding logic diagram is shown in Figure 4).

3.1. Regularity Extraction Techniques

The task of regularity extraction is to identify a set of templates and their instances from the RTL description of the circuit (*template generation step*), and then to cover the given circuit by a subset of these templates (*circuit covering step*), where the objective is to use large templates that have a large number of instances. Figure 5 illustrates a circuit cover with four templates, where template T1 has six instances, T2 has three instances, and so on. The extraction step involves a tradeoff, since a large template usually has a few instances, while a small template has a high number of instances. Note that the template composed of T2 followed by T1 has only three instances, compared to six instances of T1. Usually, a large template implies a better optimization of area and performance, while a template with more instances requires less design effort, assuming a template is synthesized only once for all its instances.

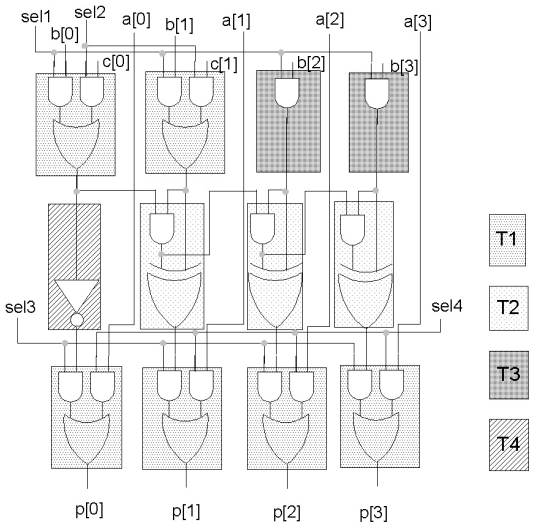

Figure 4: Logic diagram of the circuit of figure 3; the four templates shown here form a circuit cover

Several techniques for extraction of functional regularity have been proposed in the literature [3, 4, 8, 9, 10, 11, 12]. Most of these techniques focus on covering a circuit by templates, assuming that a library of templates is provided by the user. Very few techniques address the problem of generating a good set of templates. Given a library of templates, Corazao *et al.* [8, 11] address the problem of mapping a circuit described at a behavioral level using templates from the target library. Their approach addresses several key subproblems, such as finding complete as well as partial matches of a template and selecting a good set of templates to optimize the clock period. Rao and Kurdahi [12] represent the input circuit as well as templates from the given library by strings, and they use a string matching algorithm to find all instances of the template in the circuit. These authors present heuristics to generate a set of templates; the final cover is highly sensitive to these templates. Odawara *et al.* [9] present a methodology to identify structural regularity in highly regular datapaths. In their method, latches driven by the same control signals as initial templates are chosen and used to grow larger templates. Odawara's approach identifies one-dimensional regularity in terms of bit-slices of the datapath. Other approaches by Nijssen *et al*. [10] and Arikati *et al.* [3] extend Odawara's methodology to identify bit slices as well as stages of datapath circuits. These structural methods perform well for highly regular circuits, but might not work for circuits with a mix of datapath and control logic. A problem similar to regularity extraction is technology mapping, where the input circuit is covered by cells (templates) from a given library. Keutzer [7] proposed partitioning the circuit into rooted trees and then mapping the trees using library cells, by using dynamic programming. All the abovementioned techniques address the problem of covering a circuit by templates, where the templates are either provided by the user or generated in an ad hoc manner. None of these techniques deal with the systematic generation of a set of templates for a given circuit.

We have designed an efficient and robust approach for extraction of functional regularity [13, 14], where the set of all possible templates is generated automatically for the input circuit under two simplifying, yet practical assumptions: (a) only maximal templates are considered, where a template is maximal if and only if all its instances are *not* entirely covered by instances of another template, and (b) input permutations of gates in the RTL description are ignored. The number of templates is reduced to within V^2, where V is the number of components in the circuit. We have demonstrated that a wide range of efficient covers are obtained for various benchmarks from the set of templates generated by our approach [13]. Since a sufficiently large set of templates is generated, and the binate covering problem is inherently difficult [5], we employ simple and efficient heuristics to cover the circuit. Our approach recursively selects a template from the complete set of templates, based on one of the following heuristics, and deletes all its non-overlapping instances from the input circuit, until the entire circuit is covered.

(a) Largest-fit-first (LFF) heuristic: select the template with the maximum area, where the area of every component is given.

(b) Most-frequent-fit-first (MFF) heuristic: select the template with the maximum number of instances.

These two heuristics give different covers; other heuristics can be used to generate a range of covers from which the designer can choose the most desirable cover. In fact, we can represent the set of covers by a template hierarchy, where regularity among different templates is recursively extracted [14]. In the event that a template is specified by the designer, using our approach, all its instances can be generated and used for finding a cover. Thus, a cover, which is a mix of automatically extracted and user-specified templates, can be generated. (We have filed a patent on our regularity extraction approach [15]).

3.2. Vector Identification

So far, we have generated templates using functional regularity [13] without accounting for the circuit structure in terms of the interconnections among the template instances. As a result, the templates do not directly correlate with the datapath vectors. For example, the six instances of template T1 in Figure 4 should belong to two different vectors. (Here, a vector is defined as a set of template instances that are grouped together for subsequent synthesis and layout stages.) We now consider structural regularity to transform templates into datapath vectors [13]. We explain the steps of vector identification using the example of Figure 4; the resulting vectors are shown in Figure 5.

- *Simple vectors*: The instances of a template are partitioned into vectors, which we call simple vectors. For example, the template T1 of Figure 4 is partitioned into two simple vectors, SV1 with two instances and SV2 with four instances. The remaining templates result in a single simple vector each.
- *Composite vectors*: Simple vectors of different templates are grouped, if possible, to form composite vectors. For example, simple vector SV1 of template T1 is grouped with the simple vector of template T3 to form a composite vector V1 (see Figure 5).

The resulting vectors of the template cover of Figure 4 are shown in Figure 5. We use a set of efficient heuristics to group template instances to form simple or composite vectors. These heuristics are listed below.

1. *Control/data inputs*: The input signals of template instances are classified as control or data from the HDL description of Figure 3, e.g., *sel1* is a control signal, while *a[0]* is a data signal. The instances with the same control inputs and similar data inputs are grouped together.
2. *Output signal name*: The instances whose outputs drive the same bus are grouped together. For example, the instances of templates T2 and T4 are grouped together, since their outputs form the bus *g[3:0]*.
3. *Circuit topology*: Two template instances are grouped in the same vector, only if one of them is not in the transitive fanin of another. This heuristic will ensure that the template T1 (Figure 4) would be partitioned into at least two simple vectors, since two of its instances are in the transitive fanin of two other instances.

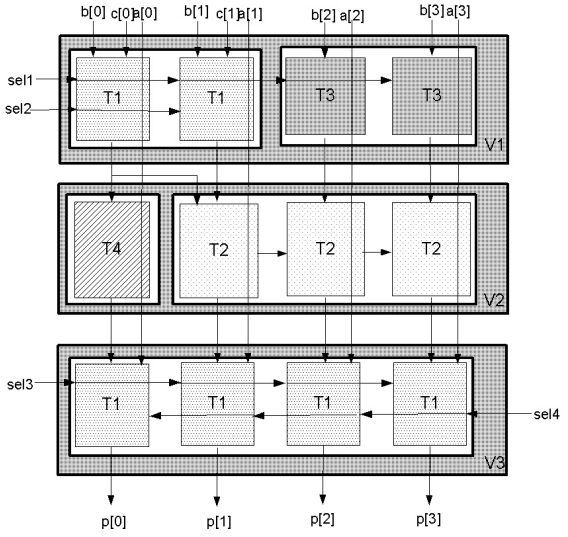

Figure 5: Schematic of example circuit obtained after forming vectors from the templates of Figure 4

3.3. Schematic Generation

A schematic of the datapath circuit is generated using the vectors identified earlier and the control/data assignment to the signals. The schematic for the example circuit is shown in Figure 5. The schematic is essential to allow designers to control the design process: (a) they can get a much better understanding of the circuit than they could from the HDL description; (b) they can modify the design hierarchy and floorplan by merging/breaking templates or vectors, changing the control/data orientation of signals, and modifying the order of vectors. An example of such a modification is merging templates T2 and T1 to form a larger template with three instances, which might lead to better optimization during subsequent steps.

3.4. Technology Mapping

The input to technology mapping is the set of datapath vectors and the I/O timing requirements in terms of input arrival times and output loads. The partition of

the circuit into vectors (or underlying templates) allows the designer to select a desired technology for each template independently. Currently, our synthesis flow assumes that the mapping of templates is performed manually, which can be easily automated due to the small size of templates. We explain several choices for mapping of templates.

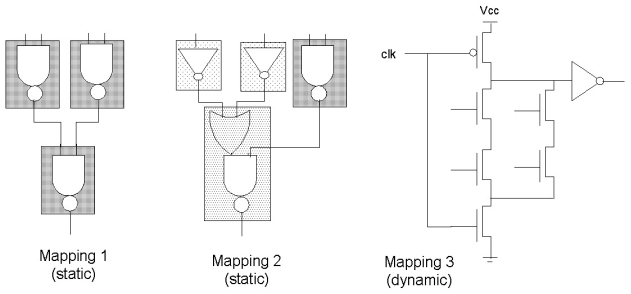

Figure 6: Several mappings of template T1 (Figure 4) to static and dynamic logic

- *Static logic*: The traditional approach of mapping a circuit to static logic first decomposes the circuit into smaller directed-acyclic graphs (DAGs) [5, 7] and then independently maps each DAG to the specified library of static cells. In our case, the templates are small enough to be mapped directly without any decomposition. Figure 6 illustrates two mappings of template T1 (Figure 4) to static logic. Here, mapping 2 is suitable for template T1 in vector V3 of Figure 5, since one of the data input signals arrives later than the other. On the other hand, mapping 1 is suitable for T1 in vector V1 (Figure 5). Thus, depending on the template usage in the circuit, we might have to use several mappings of a template.
- *Dynamic logic:* A template can be mapped to dynamic logic to achieve better timing; however, noise-related issues have to be considered, such as the length of the input and output signals of the template. Figure 4 also shows a mapping of template T1 (Figure 4) to dynamic logic. We are looking into automating the mapping of templates to dynamic logic.
- *Macro cells:* Datapath circuits employ commonly occurring logic blocks, such as incrementers, adders, shifters, etc. A library of various mappings of these specialized datapath blocks for a range of area, performance, and power values will be required. For example, the incrementer in the example circuit of Figures 3 - 5 can be replaced by one of its mapped versions prior to extracting regularity from the HDL description; our synthesis flow would then result in vectors V1 and V3 shown in Figure 5, while V2 would correspond to a macro cell.

3.5. Gate Sizing

Once all the templates of a circuit are mapped to the desired technology, every gate is sized to satisfy the output load requirements. The output load capacitance of a gate comprises the following components:

1. *Gate capacitance*: The capacitance values of the gates driven by this particular gate are available after the technology mapping step.
2. *Diffusion capacitance*: The diffusion capacitance of the gate is also known after technology mapping.
3. *Interconnect capacitance*: The capacitance is available only after the post-synthesis steps of floorplanning and RC estimation. Therefore, interconnect capacitance is used only in the gate-sizing step in the subsequent design iterations.
4. *Primary output load capacitance*: The load capacitance is already specified for the primary outputs of the circuit.

The sizing of the gates of the mapped circuit is performed starting from the primary outputs and traversing back to the primary inputs, where the output load requirement is satisfied for every gate encountered. If there are loops in the circuit, then the gate sizes will take a few iterations to converge.

Different instances of a template mapping will be sized differently depending on the output load requirements. In general, a template with multiple instances can have several mappings, where each mapping can have several different gate sizes.

Gate sizing is performed again after the interconnect capacitance values are obtained from the floorplanning and RC estimation steps.

3.6. Results

While the steps of technology mapping and sizing are still under development, we have implemented prototypes for regularity extraction and vector identification. We list below the results of regularity extraction and vector identification on two datapath blocks in terms of the number of templates, vectors, and their instances.

Ckt.	No. of components	No. of templates (instances)	Regularity index	No. of datapath vectors
Block1	464	9(400)	2%	7
Block2	318	17(124)	12%	9

We have defined an index, called a *regularity index*, to evaluate the results of regularity extraction [13]. The regularity index is defined as the percentage of the number of logic components in all the templates to the total number of logic components in the circuit. The regularity index correlates to the reduction in the design effort, assuming that a template is not synthesized multiple times for its multiple instances.

IV. Datapath Block Floorplanning and Placement

4.1. Objectives of Layout Planning

Layout planning of datapath blocks is used to obtain early estimates for block area and timing of critical signals. The layout-based estimates are used during the circuit design stage to carry out more accurate circuit simulations and design of datapath circuit schematics. Layout planning support should provide the following:

- speed and high interactivity (to enable what-if analysis)
- reasonable estimates for area and parasitics
- tradeoffs between accuracy and tool performance

We have developed a set of tools, based on experience from a recent microprocessor design project. These tools provide a user with the means to estimate the layout area of a datapath block and the interconnect parasitics from which quick timing analysis can be performed. The designers can also estimate interconnect parasitics derived from minimum spanning trees for rough estimates and actual global routes for more accurate estimates.

4.2. Tasks in Layout Planning

The inputs to the tools are top-down block pin interface, user-defined placement hints, and the netlist (which may be incomplete). The tool provides a means to visually see and edit the placement and change the netlist. The netlist is modified if a cell used in the block is changed because of the need for higher drive strength or other interconnect optimization requirements. The key functions performed in the layout planning stage for interconnect optimization are as follows:

- cell area estimation and interface design
- identification of vectors and rows
- placement of cell instances (as part of vectors)
- global routing and congestion analysis
- parasitic estimation and timing analysis

A key feature of the datapath layout planning is the layout modeling. Due to the frequent occurrence of multiple instantiations of a logic cell in datapath blocks, an entity called a vector is created to represent a group of instances, and layout editing on these groups of instances is supported. Further, it is also observed that the contents of a stage in the circuit design are placed in a row and that the contents of a bit-slice are placed in a column. Thus the complete layout plan is modeled as a matrix. Commands are then provided to move, delete, and/or create vectors, rows, matrices, etc. This method of layout modeling helps ensure regularity in the placement of cells in the layout plan.

After a reasonable placement has been determined, a designer will then estimate interconnect parasitics. Location of interface ports of the cell can also be planned to enable better routing [16, 17]. The interface planning can be carried out using Track Share Analysis (TSA) or global routing. Based on the results of the global routing or the TSA, the interface terminals (pins and ports) of the cells are placed at appropriate locations, and the net length estimation process proceeds.

An interactive graphical user environment has been developed to support this layout planning process. This environment also provides other features. A user can plan for routing space and analyze routing congestion information, which is derived from global routing. Based on the congestion analysis, the user can manually adjust the placement and plan out for area. The environment also provides net visualization and editing functionality to interactively optimize the interconnect delay. The overall design flow for layout planning is shown in Figure 7.

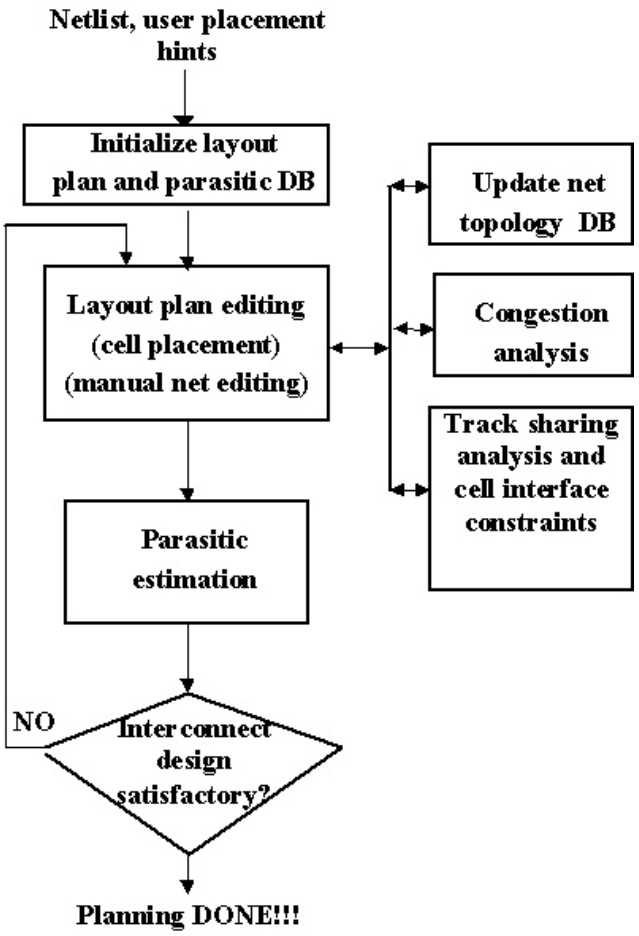

Figure 7: Layout planning design flow

4.3. Benefits of Layout Planning

From a recent Intel® microprocessor design project, effort analysis indicated that when early layout planning is carried out, the amount of re-design and rework required is reduced by approximately half. This benefit is expected to be even more significant in the future when we have more stringent and complex design requirements.

4.4. The Challenges in Datapath Auto Placement

Placement is a very critical stage in the datapath layout design flow as it can make or break regularity, area, and timing specifications. If a designer starts out with a bad placement, it is extremely difficult for the router to make up for it.

The main difference between datapath placement and Random Logic Synthesis (RLS) placement is the need to maintain regularity and hierarchy. Maintaining regularity in datapath placement offers several advantages. The circuit designer can rely on the regularity to use his intuition about critical paths. Regular layout tends to be more dense because of the reduction in the number of jogs/bends and because the designers can spend more time on optimizing one bit-slice. Regularity and hierarchy in layout are also very useful in reducing ECO time.

Timing and area constraints also tend to be much more critical for datapath blocks than RLS blocks. Unlike RLS blocks, datapath blocks are often made from custom designed cells that don't come with all the timing characterization data. This poses additional challenges for timing-driven placement algorithms.

Not all datapaths are fully regular, and they show differing amounts of irregularity, something the datapath auto-placement algorithm must contend with.

Traditional auto placement techniques, based on mathematical programming (usually with a quadratic objective function) or simulated annealing, can be modified to deal with the unique requirements of datapath placement with varying degrees of success. Techniques based on quadratic programming tend to be faster, but the rigid formulation makes it difficult to directly model the regularity requirements.

V. RC Estimation

Increased use of noise sensitive dynamic circuits, lower supply voltages, and increasing current density have made more extensive interconnect analyses a requirement in the design process. Such effects must be modeled at all stages of the design process. Estimation of the effects of interconnects and device parasitics must be accurate and consistent at all stages of the design in order to avoid unnecessary design iterations. Accurate parasitic estimation in the datapath design flow depends on both the prediction of the physical properties of the interconnects and devices (topology, routing layers, density, device layout, etc.) and on the accurate modeling of the parasitic effects of the devices and routing.

5.1. Layout Estimation Techniques

A wide range of layout estimation techniques are in use in design tools, ranging from wire length estimation to detailed net topology estimation. Such techniques are based on a set of rules, such as default routing layers, widths, and spacing, and on net topology generation algorithms such as a minimum spanning tree or Steiner tree (minimum length routing tree with horizontal and vertical wires). Some estimates may account for metal density or routing congestion constraints. The accuracy of layout estimation is dependent on the state of the design data. Estimated layout based on a globally routed floorplan may be very close to the final detailed routing, while schematic-

based estimates using little physical design data may correspond poorly with the final design. Thus, the quality of the estimated layout is highly dependent on how well the datapath design tools provide an early estimation of the physical design.

5.2. Parasitics Modeling Techniques

The modeling of process-related effects is a fairly mature field, with a wide range of tools, models, and techniques in use. Models range from empirical, easy to evaluate equations [20], to computationally intensive field solvers [19]. A wide range of parasitics' modeling tools are available both commercially and from universities. Commercially available tools provide reasonable accuracy (within 10% of field solvers) on large designs, and field solver accuracy is possible on a per-net basis [21]. Most commercial tools handle only post-layout parasitic extraction and are suitable only for final verification of designs. Many analysis tools and physical design tools (such as circuit analysis tools or global routers) have built-in parasitic estimation capability to estimate the effect of interconnect parasitics, but such tools cover only a part of the design process. The models used by these tools may not make use of all available design data, and inconsistency in the parasitics models used by different tools may result in poor convergence of the design and increased design cycle time. In addition, the built-in estimation may not accurately model cross capacitance and may not easily extend to new types of analysis required in the design flow.

5.3. Parasitic Estimation

Our work on parasitic estimation in the datapath design process focuses on accurate, consistent parasitic estimation at all stages of the design. The first and perhaps most important element in the accurate estimation of parasitics is the datapath design flow itself, particularly the close interaction and sharing of design data between the tools in the flow. The next element is the flexibility of the parasitic estimation tool to handle design data at all stages of completion, and the ability to support the wide range of constraints and assumptions required at each stage of the design. An extensive net specification system is an integral part of the design tool suite, providing designers the ability to specify a wide range of properties on the nets in the design. These net specifications are used by the parasitic estimation capability to ensure that the parasitic estimates accurately reflect the designer's intentions.

The parasitic estimation capability works by using all available design data to build a description of each of the nets in the design as well as the environment surrounding the nets. The estimator is based on a common representation of the layout and connectivity data. Design data from various tools in the datapath design flow are translated into this representation. Before the final stage of the design, when the layout is complete, the data for the nets will be incomplete. For example, in the floorplanning stages, the net's routing topology will not be available. Using a range of assumptions, the missing net data will be estimated. These assumptions may be tuned to match a particular layout design style. A key advance over existing parasitic estimation tools is that we are able to make use of any real layout data that exists. Estimated layout is used only when necessary to complete a net's representation. Since even drawn layout may not represent the final design, the estimator provides the capability to ignore any existing layout and replace it with estimated layout.

Next, the appropriate model is used to estimate the parasitics for each net. In our datapath design flow, the parasitic estimation tool is able to make use of a mix of input data sources and assumptions. We have developed a consistent set of models of varying accuracy that are built into the estimation tool. These models estimate interconnect and device resistance and capacitance, including cross capacitance. The estimator applies the appropriate model based on the source of the input data. The model used depends on the confidence of the original design data. Higher accuracy models are used when there is higher confidence in the design data. For example, an estimation based on a floorplan for a preliminary schematic need not use a high-accuracy model since the design is likely to change, while in the later stages of the design when much of the layout is complete, a high-accuracy model is needed to estimate cross capacitance between the nets.

The flexibility provided by the parasitic estimator allows the same tool to be used at all stages of the datapath design and helps ensure consistent results of the analyses at each stage of the design. It should be emphasized that the effectiveness of the parasitic estimator is dependent on the consistency of the results of each of the stages of the design process in the sense that the design at any stage provides an accurate estimation of the next stage and a reasonable early estimate of the final design. As shown in the other sections of this paper, this will be the case.

5.4. Results

Our initial results have shown that the parasitic estimator provides superior accuracy compared to the estimators used in existing point tools in the current datapath design flow. The benefits of the estimator will increase further when it is consistently used in the complete datapath design flow.

VI. Layout Cell Generation

The research in the field of cell synthesis was started more than 15 years ago [42]. Most of this research has focused on the generation of so-called 1-dimensional (1D) layouts when transistors are arranged in a linear fashion to minimize the number of diffusion breaks. First approximated algorithm for this layout style has been suggested by Uehara and VanCleemput [42]. Maziasz and Hayes [37] presented the first optimal algorithm.

Unfortunately 1D layout style is suitable only for small cells with fully complementary non-ratioed series-parallel CMOS circuits. Multiple attempts to extend this style have been made to handle more complicated circuit structures [23, 24, 31, 33, 36, 39].

Analysis of manually drawn layouts shows that "two-dimensional" (2D) layouts must be generated. Various approaches have been taken to address this problem [26, 27, 28, 29, 30, 40, 41, 43].

Though some of these leaf-cell layout systems have been applied successfully in ASIC flows, no commercially available system today has the capabilities to address the requirements of a custom design flow such as microprocessor design, where layout cell design involves a number of complex requirements. As chip designs approach GHz frequencies, reliability verification (RV) constraints, arising from the electro-migration and self-heat phenomenon, have also proven to be a critical factor in the generation of leaf-cell layouts.

6.1. Feature Requirements

A cell layout generation system is being looked into by us [44]. The system has to enable automated layout generation to produce cells that are optimized for various constraints such as density, performance, RV, and power. Its goal is to increase cell design productivity.

The system should include the following features:

- Ability to handle several hundred devices with various types of top-down constraints such as pre-routes, keep-out regions, pin/port preferred locations, etc.
- Easy configuration for various design domains (standard cell libraries, datapath bit-cells and bit-slice synthesis, custom cell design, etc.) and different circuit design methodology. Users should be able to define their own cell architecture rules.
- True 2D placement with RV constraints that allows simultaneous placement of cell instances and devices.
- Automatic stack and/or device-based legging with optional user control.
- Incremental area routing.
- Incremental compaction with different types of gridding constraints.
- Link with schematic editor.
- Powerful ECO mode / family generation / process migration capabilities.
- On-line RV estimation, DRC, and OpenChecker.
- Integrated with a layout editing system to allow manual intervention at any stage, ranging from push-button mode (fully automatic) to an interactive mode with unlimited manual intervention.

6.2. System Overview

In order to implement this layout generation system, five main components are required: a placer, a router, an RV analyzer, a compactor, and a family generator and change manager. A layout generation flow can be built around these five components (Figure 8). This flow can either be fully automated, or it can be guided and enhanced by a layout designer wherever required.

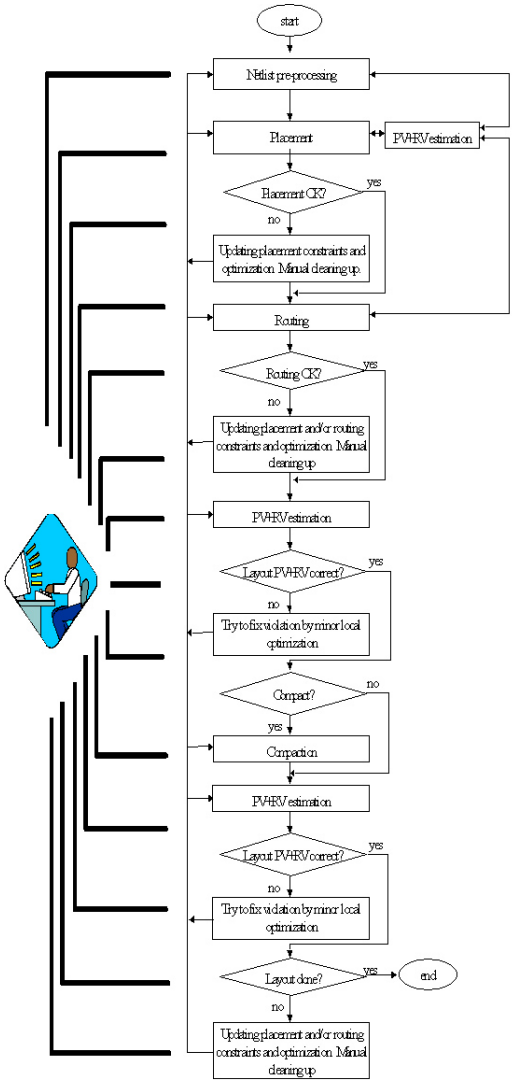

Figure 8: ***Layout synthesis flow***

6.2.1. Placer

In the generation of a layout for any schematic, a large amount of effort is spent in transferring the netlist to the layout editor, ordering the devices, and then determining the best placement for those devices. The placer has to handle a capacity for several hundred devices and be able to do two-dimensional device placement. It should also have provisions for top-down constraints, RV constraints, and an incremental placement capability.

6.2.2. Router

Once the devices have been placed, the connections between them have to be made. These are done by the routing module. Manual pre-routing of critical nets is allowed, and often encouraged, to meet strict timing or port location guidelines. The router interacts with the RV estimator to deduce the optimal routing shapes for critical nets based on given RV constraints. Once the optimal routing topologies have been determined, the actual routing itself is done by a detailed router. To improve routing quality, the router module refines the placement based on congestion analysis.

6.2.3. RV Analyzer

At different stages of the work flow, RV estimations are required to produce layouts that are optimized for reliability constraints. The RV estimator is based on worst-case current analysis through static modeling of current switching. It has a built-in current-solving engine that traverses through nets to compute worst-case interconnect currents from the switching of the device stacks. Based on the results of the analysis, the module identifies objects that are electro-migration and self-heat limited. The RV analysis can potentially lead to a re-ordering of devices, a change in routing topology, or a change in wire and via geometries.

6.2.4. Compactor

This is used to compact the area and resolve design rule violations, as well as for putting pins on grid for supporting the routing flow at the next level of design hierarchy. It can be configured by a wide range of options to support a specific working flow.

6.2.5. Family Generator and Change Manager

While generating cell libraries, several cells of similar topologies need to be created, the differences usually being ones of device sizing, with minor changes in schematic, legging, etc. The same situation is also encountered if the schematics are revised after the layout has been done. Since such changes don't modify the fundamental layout topology, we can generate subsequent cells from a starting prototype or template. This is done by creating a mapping between the netlists of the template and the desired cell, followed by resizing, and adding or deleting devices or legs as necessary. Using the family generation module, the layout designer only has to lay out a couple of representative cells of a cell family. The layouts for all the other members of the same cell family are then generated automatically. This feature can also be used for process migration.

Each of the above steps are independent of each other: for example, the devices may be manually placed and then automatically routed, or, cells drawn manually may be used as templates for family generation, and so on. This ensures improved layout design productivity without compromising the layout quality.

6.3. Results

Initial usage of a prototype version of the cell generation system at Intel Corporation shows significant productivity improvement over manual design for various kinds of cells, while meeting all layout quality requirements such as density, reliability, power, and timing.

VII. Future Challenges

It is expected that with continuous process technology advancement and the growing need for higher performance chips, the problems in datapath design will continue to increase and become more complex. Granted, not all problems are known or understood at this time. There are a number of problems that we are dealing with currently, which will get much worse in the future. They are as follows:

- *Handling of coupling noise problems (primarily due to capacitive coupling).* A substantial amount of effort is currently required to verify and correct the design to ensure correct silicon behavior. Techniques to generate correct-by-construction noise problem-free circuit and layout are essential.
- *Timing analysis to include effects of noise (power noise, capacitive coupling, and inductive coupling)*. Guard banding (in timing analysis cycle time or interconnect capacitance) is often used to account for the effect of noise. However over-conservatism would result if the guard banding is done for the worst-case scenario. If some statistical averages are used in guard banding, there might be serious escapes, which can cause problems in silicon. Hence, it is necessary to have the ability to include the effects of noise accurately in timing analysis.
- *New circuit techniques*. Traditional static CMOS and domino logic circuits have worked well so far. However, with the continuous decrease in power supply voltage and the increased demand in chip performance, new circuit design styles have to be investigated to achieve a better delay-power product and to meet other design requirements.

VIII. Conclusion

In this paper, we have presented the challenges in datapath design and our ideas to meet these challenges through datapath logic synthesis, layout planning, interconnect RC estimation, and layout cell generation. We believe that datapath design requires substantially more automation to be able to meet future requirements: "system on a chip" and demand for higher performance and deep sub-micron geometries. We hope that this paper stimulates more interest in both academic and commercial CAD arenas to tackle the problems in high-performance datapath design.

Acknowledgments

We thank Nagbhushan Veerapaneni who contributed to the paper in the area of auto placement. We also thank Marian Lacey, Mysore Sriram, Bharat Bhushan, and Lin Chao who reviewed this paper and gave valuable feedback.

References

[1] D.E. Hoffman, "Deep Submicron Design Techniques for the 500MHz IBM S/390 G5 Custom Microprocessor," Proc. of ICCD 1998.

[2] S. Posluszny, "Design Methdology for a 1.0 GHz Microprocessor," Proc. of ICCD 1998.

[3] S. R. Arikati and R. Varadarajan, "A signature based approach to regularity extraction," *Proc. of ICCAD*, November 1997, pp. 542-545.

[4] M. Hirsch and D. Siewiorek, "Automatically extracting structures from a logical design," *Proc. of ICCAD*, November 1988, pp. 456-459.

[5] G. de Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw Hill, New York, 1990.

[6] E. Detjens, *et al*., "Technology mapping in MIS," *Proc. of ICCAD*, November, 1987, pp. 116-119.

[7] K. Keutzer, Dagon, "Technology binding and local optimization by DAG matching," *Proc. of DAC*, June 1987.

[8] M. R.Corazao, *et al*., "Performance optimization using template matching for datapath-intensive high-level synthesis," *IEEE Trans. on CAD*, 15(8), August 1996, pp. 877-887.

[9] G. Odawara, *et al*., "Partitioning and placement technique for CMOS gate arrays," *IEEE Trans. on CAD*, May 1987, pp. 355-363.

[10] R.X.T. Nijseen, and C. A. J. van Eijk, "Regular layout generation of logically optimized datapaths," *Proc. of ISPD*, 1997, pp. 42-47.

[11] J. M. Rabaey, *et al,*. "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test of Computers*, June 1991, pp. 40-51.

[12] D. S. Rao and F. J. Kurdahi, "On clustering for maximal regularity extraction," *IEEE Trans. on CAD*, 12(8), August 1993, pp. 1198-1208.

[13] A. Chowdhary, *et al*., "A general approach for regularity extraction in datapath circuits," *Proc. of ICCAD*, November 1998, pp. 332-339.

[14] A. Chowdhary, *et al*., "Extraction of functional regularity in datapath circuits," *IEEE Trans. on CAD*, submitted November 1998.

[15] A. Chowdhary, *et al*., "A systematic approach for regularity extraction," *U.S. Patent*, filed November 7, 1998.

[16] B. Krishna, C.Y.R. Chen, and N. Sehgal, "Technique for Planning of Terminal locations of Leaf Cells in Cell-Based Design," *Proc. 11th International Conference On VLSI Design*, pp. 53-58, 1998.

[17] Amnon Baron Cohen, Michael Shechory, "Track Assignment in the Pathway Datapath Layout Assembler," *Digest of Technical Papers 1991 IEEE International Conference on Computer-Aided Design*, pp. 102-105, 1991.

[18] J. Cohn, L. Pillage, and I. Young, "Tutorial 4: Digital Circuit Interconnect: Issues, Models, Analysis, and Design," *IEEE/ACM International Conference on CAD-94.*

[19] J. R. Phillips and J. White, "A Precorrected-FFT Method for Capacitance Extraction of Complicated 3-D Structures," *Proc. ICCAD-94*, pp. 268-271.

[20] N. Delorme, M. Belleville, and J. Chilo, "Inductance and Capacitance Analysis Formulas for VLSI Interconnects," *Electronics Letters*, vol. 32, no. 11, May 1996.

[21] Y.L. Le Coz , R.B. Iverson, H.J. Greub, P.M. Campbell, and J.F. McDonald, "Application of a Floating-Random-Walk Algorithm for Extracting Capacitances in a Realistic HBT Fast-Risc RAM Cell," *Proc. 11th International VLSI Multilevel Interconnection Conference*, Santa Clara, CA, pp. 342-4, June 1994.

[22] D.G. Baltus and J.Allen, "SOLO: A generator of efficient layouts from optimized MOS circuit schematics," *Proc. 25th ACM/IEEE Design Automation Conferenec,* pp. 445-452, June 1988.

[23] B.Basaran, "Optimal Diffusion Sharing in Digital and Analog CMOS Layout," Ph.D. Dissertation, Carnegie Mellon University, CMU Report No. CMUCAD-97-21, May 1997.

[24] J.Burns and J.Feldman, "C5M: A Control Logic Layout Synthesis System for High-Performance Microprocessors," *Proc. ISPD'97*, pp. 110-115.

[25] C.C.Chen and S.L.Chow, "The layout synthesizer: An automatic netlist-to-layout system," *Proc. 26th ACM/IEEE Design Automation Conference,* pp. 232-238, June 1989.

[26] S.Chow, H.Chang, J.Lam, and Y.Liao, "The Layout Synthesizer: An Automatic Block Generation System," *Proc. CICC* 1992, pp. 11.1.1-11.1.4.

[27] J.Cohn, D.Garrod, R.Rutenba, and L.R.Carley, *Analog Device-Level Layout Automation*, Kluwer Academic Publishers, Boston MA, 1994.

[28] M.Fukui, N.Shinomiya, and T.Akino, "A New Layout Synthesis for Leaf Cell Design," *Proc. 1995 ASP-DAC*, pp. 259-263.

[29] A.Gupta and J.Hayes, "Width Minimization of Two-Dimensional CMOS Cells Using Integer Linear Programming," *Proc. ICCAD* 1996, pp.660-667.

[30] A.Gupta and J.Hayes, "CLIP: An Optimizing Layout Generator for Two-Dimensional CMOS Cells," *Proc. 34th DAC* 1997, pp.452-4557.

[31] A. Gupta and J. Hayes, "Optimal 2-D Cell Layout with Integrated Transistor Folding," *Proc. ICCAD* 1998, pp.128-135.

[32] M. Guruswamy, R. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez, and L. Jones "CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries," *Proc DAC'97*, pp. 327-332.

[33] Y-C. Hsieh, C-Y. Hwang, Y-L. Lin, and Y-C. Hsu, "LiB: A CMOS cell compiler," *IEEE Transactions on Computer Aided Design,* Vol. 10, pp. 994-1005, August 1991.

[34] Chi Yi Hwang, Yung-Ching Hsieh, Youn-Long Lin, and Yu-Chin Hsu, "An Efficient Layout Style for Two-Metal CMOS Leaf Cells and its Automatic Synthesis," *IEEE Transactions on Computer Aided Design*, Vol. 12, pp. 410-423, March 1993.

[35] M. Lefebvre, C. Chan, and G. Martin, "Transistor placement and interconnect algorithms for leaf cell synthesis," *EDAC-90*, pp. 119-123, 1990.

[36] M. Lefebvre and D. Scoll, "PicasoII: A CMOS Leaf Cell Synthesis System," *Proc. 1992 MCNC Intl. Workshop on Layout Synthesis*, Vol. 2, pp. 207-219.

[37] R.Maziasz and J.Hayes, "*Layout Minimization of CMOS Cells*," Kluwer Academic Publishers, Boston, 1992.

[38] C.L. Ong, J.T. Li and C.Y. Lo, "GENAC: An automatic cell synthesis tool," *Proc. 26th ACM/*

IEEE Design Automation Conference, pp. 239-244, June 1989.

[39] C.Poirier, "Excellerator: Custom CMOS Leaf Cell Layout Generation, " *IEEE Trans. on CAD, 8(7)*, July 1989, pp. 744-755.

[40] S. Saika, M. Fukui, N. Shinomiya, and T. Akino, "A Two-Dimensional Transistor Placement Algorithm for Cell Synthesis and its Application to Standard Cells," *IEICE Trans. Fund., E80-A(10)*, Oct.1997, pp. 1883-1891.

[41] K. Tani, K. Izumi, M. Kashimura, T. Matsuda, and T. Fujii, "Two-Dimensional Layout Synthesis for Large-Scale CMOS Circuits," *Proc ICCAD 1992*, pp.490-493.

[42] T. Uehara and W.M. VanCleemput, "Optimal Layout of CMOS Functional Arrays*," IEEE Transactions on Computers, C-30(5),* May 1981, pp. 305-312.

[43] H. Xia, M. Lefebvre, and D. Vinke, "Optimization-Based Placement Algorithm for BiCMOS Leaf Cell Generation," *IEEE J.Solid State Circ., 29(10)*, October 1994, pp. 1227-1237.

[44] B.Basaran, K.Ganesh, A.Levin, R.Lau, M.McCoo, S.Rangarajan, and N.Sehgal, "GeneSys - A Layout Synthesis System for GHz VLSI Designs," Proc. 12th International Conference on VLSI Design, January 1999, pp. 458-452.

Authors' Biographies

Tim Chan is currently a CAD tool system architect at Intel. He received his B.Sc. and M. Phil. in electrical engineering from the University of Hong Kong in 1980 and 1984 respectively. He joined Intel in 1990 as a senior VLSI designer. His technical interests include high-speed circuit design, microprocessor design methodology, and logic synthesis. His e-mail is tim.w.chan@intel.com.

Amit Chowdhary is a senior CAD engineer at Intel. He received his Ph.D. in computer science and engineering from the University of Michigan, Ann Arbor in 1997. He joined Intel in 1997 and is working on the datapath automation project in Design Technology (Microprocessors Products Group). His technical interests include high-level and logic synthesis, technology mapping, and timing analysis. His e-mail is amit.chowdhary@intel.com.

Bharat Krishna is a senior CAD engineer in the Microprocessors Product Group at Intel. He received an M.S. in computer engineering from Syracuse University in 1994 and a B.Sc. in electrical engineering from the University of Khartoum, Sudan in 1991. He has been working in Intel since 1995, and he is the project leader for the layout planner tool. His interests include datapath layout automation and VLSI routing. His e-mail is bharat.krishna@intel.com.

Artour Levin is a staff CAD engineer at Intel. He received his MS in Math from Belarus State University in 1986 and Ph.D. in Computer Science from the same University in 1990. He joined Intel in 1995 and is working on CAD tool and methodology development in the Microprocessors Product Group. His interests include discrete mathematics, combinatorial optimization, CAD algorithms and design methodology. His e-mail address is alevin@scdt.intel.com.

Gary Meeker Jr. is a senior CAD engineer at Intel. He received his BSEE from Carnegie Mellon University in 1990 and his MSEE from UC Berkeley in 1994. He joined Intel in 1994 and is working on CAD tool and methodology development in the Microprocessors Product Group. His interests include parasitic extraction and estimation tools, algorithms, and models. His e-mail is gmeeker@scdt.intel.com.

Naresh Sehgal is currently managing the Datapath Tool Development Group for next-generation processor design at Intel. He received his B.S. in EE from Punjab Engineering College in India, followed by an M.S. and Ph.D. in computer engineering from Syracuse University, NY. Naresh has been with Intel since 1988, and his research interests include CAD algorithms and design methodology. His e-mail is naresh.sehgal@intel.com.